OMNIMON! (TM)
USER'S GUIDE

by

David Young, CDY Consulting

SYSTEM REQUIREMENTS: ATARI 800/400 Home Computer

AUTHOR'S EARNEST ENTREATY

I have done my best to offer here a quality program at a reasonable price. This is my livelihood. Please do not make copies of this program for any reason other than personal backup. Thank you.

INTRODUCTION

Greetings fellow ATARI Home Computer owner. I am sure you are just as proud of your system as I am of mine and enjoy buying accessories to extend its power and convenience. From that point of view, OMNIMON! is one of the most powerful additions you can make to your computer. Any serious ATARI owner will find it indispensable after using it for the first time.

OMNIMON! is a resident machine language monitor which, once installed, is always available to you. What that means is that you never have to load it and you can call it up no matter what program happens to be running at the time. Once running, OMNIMON! gives you complete control over your computer. This includes the ability to easily examine and modify memory or the 6502's registers, to dump data to a printer, and to read and write to the disk drive(s) without DOS. It also has a complete set of debugging tools including a disassembler, single step, and a unique JSR function for testing out subroutines. And all of these features are available to you at any time, no matter what program is running, simply by pressing SYSTEM RESET along with either the OPTION or SELECT button!

If you have been wanting to learn assembly language programming, OMNIMON! can make it a very pleasant experience. Since it is ROM resident, you can always get back to OMNIMON! even if your program hangs up in an infinite loop. (See the description of OMNIMONA, the advanced version of the monitor, for a method of recovery if the system is locked up.) You can even call OMNIMON! at critical points in your program to examine things before continuing execution. OMNIMON! is extremely user friendly so even programmers with little experience should find it very easy to use.

GETTING STARTED

After installing OMNIMON! in your computer (if you have not done so, see OMNIMON! INSTALLATION INSTRUCTIONS), you should be able to powerup your computer as usual. To enter the OMNIMON! program, hold down the OPTION key and press SYSTEM RESET. This method of entering OMNIMON! will cause a warmstart up to the point that the application program would normally be given control. Instead OMNIMON! takes control and you should see the OMNIMON! header written across the top of the screen indicating that the program is running:

```
       David Young OMNIMON! Copyright 1983

   PC NV-BDIZC ACCUM X-REG Y-REG STACK
Axxxx    xx     xx    xx    xx    xxx
```

When you are ready to exit OMNIMON!, hold down the START button and type RETURN. This will cause the warmstart to go to completion, giving control back to the application program. Notice that pressing SYSTEM RESET by itself will cause a normal warmstart. Later we will discuss another method of entering OMNIMON! which preserves the PC and CPU registers of the interrupted program.

Once you have OMNIMON! running, the first command to learn is the 'HELP' command. As is fairly standard practice in user friendly software, OMNIMON! uses '?' (RETURN) to give you a list of all the commands available. So type '?' followed by RETURN and you will see the following:

```
CPU/CHG:C
DPY/CHG:D (adr adr)
EXECUTE:E (byt)
JSR    :J adr
LINK DR:L (drive#)
PRINTER:P
RD DISK:R (sec# adr #)
SEARCH :S adr byt byt ...
TOGGLE :T
WR DISK:W (sec# adr #)
DIS/CHG:X (adr adr)
PSH STK:+ byt byt ...
POP STK:-
```

The HELP command not only provides a list of commands but also indicates the parameters each command expects. Parameters in parentheses are optional. If they are omitted, OMNIMON! will try to interpret the command in a manner convenient to you. Usually this means executing the command on the next logical memory location or sector. If you are anxious to start using OMNIMON!, you can do so immediately with just this little bit of knowledge. With a little experimentation you should have little trouble figuring out what most of the commands do. When you are ready to learn some of the more subtle features built into OMNIMON!, read the rest of this documentation.

There are a few important things to point out before proceeding:

1) All numerical input and output is done in hex.

2) Parameters are delimited by a space or other non-hex character.

3) The command being processed will be aborted if an illegal parameter is encountered or if a necessary parameter is not supplied.

4) It is not necessary to retype a command if it is already present on the screen. Just position the cursor on the same line, make changes if you wish, and type RETURN. All the normal ATARI editing commands are available.

5) The processing of most commands can be stopped by holding down the START button. This allows you to terminate a long listing, search or single step. Use CTRL-1 to temporarily halt and restart a listing.

DISPLAY MEMORY: D (start addr) (end addr)

This command is used to view data in memory in either hex or character format, depending on the current data format (see TOGGLE). In hex format the data is output to the screen as 1 or more lines of 8 hex bytes separated by spaces. In character format the data is output as 1 or more lines of 24 byte character strings. On each line, the address of the first byte precedes the data.

In either data format the letter 'A' is appended to the start of each line. This in fact represents the ALTER MEMORY command (see the following command description). The effect is that, once you have used 'D' to display part of memory, you can alter any byte(s) by simply positioning the cursor, typing the change(s), and hitting RETURN. You must type RETURN on each line that you alter for the change to take effect. Also, the current data format must match the way the data was represented on the line. One other limitation in the character mode is that a line containing the character representing $9B is not alterable past that character. The OS cannot handle a record with an imbedded $9B. If you wish to alter a line after a $9B character, redisplay the line starting just past it.

To display memory, type 'D' followed by optional start and stop addresses and then RETURN. You can display up to 32K bytes ($8000) with one command. If you omit the stop address, only a single line of data will be printed. If you omit the start address, the next logical line of data will be printed (either the next 8 or 24 bytes of memory, depending on the data format). One last convenience is that once you have used the 'D' command, OMNIMON! will default to that command if you just type RETURN. This allows you to scroll through memory by holding down the RETURN key. This default will remain in effect until one of the other 'persistent' commands (R or X) are used, at which time they will become the default.

TOGGLE DATA FORMAT: T

As mentioned previously, all numerical data is represented in hex. However, when dealing with ASCII text it is more convenient to work in character format. The TOGGLE command (T) is used to switch between hex and character format. It affects three commands: ALTER MEMORY (A), DISPLAY MEMORY (D) and SEARCH MEMORY (S). Other commands are unaffected by the current data format.

To switch data formats type 'T' (RETURN). Upon first entering OMNIMON! the data type defaults to hex.

ALTER MEMORY: A addr byte byte ...

This command is used to change 1 or more contiguous bytes of memory. You can type the change either as hex bytes (separated by spaces) or as ATASCII character strings, depending on the current data format (see TOGGLE). While it is possible to use the 'A' command by itself at any time, it is not recommended. To display the area of memory first with the 'D' command and then to position the cursor and make the change is much safer (see DISPLAY MEMORY). This way you not only verify that the memory at that location is the memory you intended to change, but also that the current data format is compatible with the data you are typing.

To use the ALTER MEMORY command, type 'A' followed by an address. Use a space to separate the address and the data and then start typing the data. If in hex format, type hex bytes delimited by spaces. If in character format, type a continuous character string. Terminate the command with RETURN. At that point the indicated changes will be made. The command line can be as long as you like or until the computer squawks. SQUAWK!

SEARCH MEMORY: S addr byte byte ...

Searching is something that computers do very well and OMNIMON! has a very nice search function that works in either hex or character mode. It will scan memory for any sequence you specify and display it in a manner similar to the DISPLAY MEMORY command every time it is found. This means you can alter any occurrence of that sequence by simply positioning the cursor, typing the change and hitting RETURN (see DISPLAY MEMORY).

To use the SEARCH MEMORY command, type 'S' followed by the address where you would like the search to begin. Then type a space followed by the search sequence. This will be hex bytes separated by spaces in hex mode or a character string in character mode (see TOGGLE). The search will begin when you hit RETURN. The search sequence can be any length up to the limit of the ATARI terminal input buffer. Even though it only takes a few seconds to search all of memory, a search can be aborted by holding down the START button.

PRINTER ON/OFF: P

If you want a hardcopy record of your OMNIMON! session, you can use the 'P' command to cause anything being output to the screen to be echoed to the printer. In character mode, inverse video characters are printed as normal video and unprintable characters are translated to dashes (-). Otherwise, everything on the screen will show up on the printer. There is even a special single step mode (see EXECUTE) that will trace through a program while outputting only to the printer and not to the screen. This is useful for programs that use the screen in modes other than GRAPHICS 0.

The 'P' command is a toggle function. Typing it once will enable output to the printer and typing it again will disable output. If the printer is not turned on or selected, the message 'I/O ERROR' will result. Care should taken if the printer is enabled while reading or writing to the disk (see READ DISK or WRITE TO DISK).

DISK INPUT/OUTPUT

Anyone who owns my disk utility DISKSCAN knows how useful it is to be able to edit raw sector data on a disk. One of my goals in designing OMNIMON! was to incorporate some of the features of DISKSCAN. Imagine, a resident mini-DISKSCAN!

Well, the end result has far exceeded my expectations. With OMNIMON! you can not only read and write individual sectors, but multiple sectors to and from anywhere in memory. And it not only works in sequential mode, but it can also follow sector links. In fact, you can read in an entire DOS file from a disk without even booting up DOS! And the frosting on the cake is that OMNIMON! works equally well in single or double density, a dream come true for the growing number of double density drive owners.

LINK/SEQ MODE & DRIVE #: L (drive#)

When you first enter OMNIMON!, the program assumes that you wish to talk to drive #1 and that the sector mode is sequential. If you wish to address other drives or follow sector links, use the LINK command to put OMNIMON! in the correct mode. The LINK command is actually two commands in one. When used by itself (without a parameter) 'L' means to toggle from sequential to linked mode or vice versa. When followed by a drive # (1-4), 'L' means to switch the drive ID to the specified drive. From that point on, all disk I/O will be directed to that drive.

To toggle between the sequential and linked sector modes, type 'L (RETURN)'.To direct disk I/O to a different drive, type 'L' followed by the drive # and RETURN.

READ DISK: R (sector#) (buffer addr) (# sectors)

The READ DISK command is one of the most powerful, user friendly functions of OMNIMON!. It can be used to read one or more sectors, either sequentially or linked, from any disk drive, single or double density. We

will start out by using the READ DISK command to read one sector at a time. You will find it behaves somewhat differently when operating on more than one sector at a time.

To read a single sector into memory type 'R' followed by the sector # and RETURN. OMNIMON! will assume a buffer address of $6000 unless you specify something different after the sector #. From that point on OMNIMON! will assume that new buffer address for subsequent disk I/O. Once you have the sector in memory you can operate on it with any of the other OMNIMON! commands including DISPLAY, ALTER, SEARCH, DISASSEMBLE, etc. One convenient feature is that, after a READ DISK command, OMNIMON! will assume the buffer address if you use 'D' or 'X' without a start address. (Try typing 'D (RETURN)' after reading a sector into memory).

Now, if you wish to read the sector which logically follows the last sector read into memory, type 'R (RETURN)'. In sequential mode, the next physical sector on the disk will be read. In linked mode, OMNIMON! will reference the sector link of the current sector to determine the next sector to read. In either case, the new sector will be read into memory at the SAME buffer address, overlaying the old sector.

NOTE: IF THE PRINTER IS ENABLED WHILE READING SINGLE SECTORS, THE SECTOR # AND BUFFER ADDRESS MUST BE SPECIFIED EACH TIME. This is because the printer and disk share the SIO DCB (Device Control Block).

Ready for a couple more examples of user friendliness? One is that the 'R' command, like 'D' and 'X', is a 'persistent' command. That means that once you use the 'R' command, OMNIMON! will default to that command if you just type RETURN. This default will remain in effect until one of the other persistent commands are used. What this means is that you can read through an entire file (or disk, if in sequential mode) by reading the first sector and then simply holding down the RETURN key. One other convenience is that OMNIMON! will not read past the end of file if it is in linked mode. Thus, if you were reading through a file as suggested above, simply hold down the RETURN key until 'EOF' is printed. At that point, the last sector of the file is at the buffer address. You are free to add something to the end of the file (perhaps an autorun vector) and then to write the sector back out with the WRITE SECTOR command.

Reading multiple sectors is somewhat different from reading single sectors. For one thing, the sector #, buffer address, and sector count must be specified each time. The other difference is that, instead of consecutive sectors overlaying each other, the buffer address is incremented between sectors so that the disk data fills memory. The exact amount by which the buffer address is incremented depends on the sector mode and the density of the drive. The effect is that in sequential mode all bytes of the sector are preserved, while in linked mode the sector links are overlayed. This is a desirable feature if you want to read an entire DOS file into memory. If you find this discussion confusing, it is recommended that you read my tutorial on ATARI DISK DATA STRUCTURES in the DISKSCAN USER'S GUIDE or the MARCH 1982 issue of COMPUTE! magazine.

An example may be of help. First, put OMNIMON! in character mode with 'T' and sequential mode with 'L'. Now type 'R 169 6000 8'. This will read in the 8 sectors of the directory into the buffer at $6000. Now type 'D' followed by several RETURNs. You will be able to read the names of the files on the disk. Choose the filename of a short file, put OMNIMON! in hex mode with 'T', and use 'D addr' to display the 5 bytes just prior to the filename. The first byte is the status while the next two are the size of the file and the next two are the start sector. Now put the program in linked mode with 'L'. Read in the entire file with 'R (start sector) 6000 (file size)'. Type 'D' and hold down the RETURN key to scroll through the data of the file.

If the file happens to be a BINARY LOAD file, a slight variation of this technique is recommended. Instead of specifying an arbitrary buffer address of $6000, look at the first sector of the file to determine the load address. Then subtract 6 bytes to account for the load vector and use that number as the buffer address. Then, after the file is loaded, it can be executed, searched, disassembled or otherwise manipulated to your hearts content. Notice that the entire program will be loaded to the correct place in memory only if there is but one load vector at the beginning of the file. OMNIMON! as a rule ignores load vectors. If you must load a file with multiple load vectors, use DOS.

Once you have a binary load file in memory you can create a boot disk by switching over to sequential mode and writing the program back out to a disk starting at sector 1. You will need to leave 6 overhead bytes at the beginning of the first sector. See the ATARI OPERATING SYSTEM USER'S MANUAL for details on the boot process. The converse of this process would be to create a binary load file from a boot record. This can be done by first booting up DOS, going to OMNIMON!, and reading in the boot record in sequential mode to a convenient place in memory. Then you would exit back to DOS and do a BINARY SAVE on that portion of memory. Then you may have to use OMNIMON! to change the load vector at the beginning of the file to make it load in at the correct place. In fact, you may have to use OMNIMON! to load the file if it loads on top of DOS. The same technique can be used to make a binary load file out of a cartridge but you will have to append a few load vectors to get the program going. For example, the following 3 load vectors should be appended to the end of BASIC: 6A 00 6A 00 90 E2 02 E3 02 F6 F3 E0 02 E1 02 00 A0.

One final convenience is that each time a single sector is read, its # is printed along with the buffer address. This also occurs when multiple sectors are read, but only when the printer is on. Thus, if you read in an entire file with the printer on, you get a sector map of that file. Now if, while inspecting the file in memory, you find that you wish to make a change to the file on disk, you can compare the buffer address to the sector map of the file to determine the sector where that piece of data resides. Then you can use 'R sec#' to fetch that sector, make the change, and use 'W sec#' to store the sector back on disk.

WRITE TO DISK: W (sector #) (buffer addr) (# sectors)

The WRITE TO DISK command allows you to write one or more sectors worth of memory out to disk. The one big difference between it and the READ DISK command is that it only works in the sequential mode. That means that it will not create a DOS file, i.e., it will create neither a directory entry nor sector links. If you do wish to create a DOS file out of memory it is best to use the BINARY SAVE option of DOS. However, it is not always possible to get DOS into memory without losing your data. In this case, OMNIMON! may be the only way save your data. If you do wish to create a DOS file out of the memory you have written to disk with OMNIMON!, use the technique described in the last section. You could also use the BINARY LOAD FILE function of DISKSCAN in the sequential mode which will pick up sectors and redeposit them as a DOS file.

IMPORTANT: Use a scratch disk when writing multiple sectors worth of memory to disk with OMNIMON!. The program pays no attention to data already on the disk and may overlay it.

The primary purpose of the WRITE TO DISK command is to support the modification of one sector at a time. A typical scenario is as follows:

> Turn the printer on, read a file into memory as described in READ DISK, and turn the printer off. Search the file to find the data to be changed. Compare the address of the data in memory to the sector map created while the file was being read in. Read that particular sector into memory with 'R sec#'. This insures that you not only have the data of the sector but also the sector link. Then alter the sector and write it back out with 'W sec#'.

Another application for the WRITE TO DISK command would be to move a block of memory from one location to another. This is accomplished by writing the data out from one buffer address and reading it back in at another. It is important to use a scratch disk for this operation or at least be very careful that you are writing to an unused portion of a disk.

Be careful when omitting the sector # because the default has already been incremented in anticipation of the next READ DISK. In fact, the only safe time to omit the sector # is when that sector is the last one of a linked file.

SEVERAL WAYS TO ENTER OMNIMON!

We have seen how to enter OMNIMON! by holding down OPTION and pressing SYSTEM RESET. This causes a normal warmstart followed by a jump subroutine (JSR) to OMNIMON!. When you exit OMNIMON! after entering it in this way (by holding down START and pressing RETURN), the warmstart goes to completion in a normal fashion. This is fine for some applications but there is another way to enter OMNIMON! which disturbs the program running as little as possible.

When you hold down SELECT and press SYSTEM RESET, the program running at the time is interrupted. However, instead of doing the entire warmstart, parts of it are skipped over so as to preserve the state of the system as much as possible. Specifically, the OS variables and the stack are left undisturbed. Usually this allows you to reenter the program by simply exiting OMNIMON! in the normal fashion. For instance, you can pop into OMNIMON! from either DOS or BASIC, execute some OMNIMON! commands, and pop back into the interrupted program almost as if you had never left it. I say 'almost' because the OS is likely to return a bogus value if it was waiting for a keystroke when it was interrupted. For that reason it is best to hit BREAK upon return to the program. Of course, if the program makes use of any graphics other than MODE 0, it is unlikely that you will be able to successfully reenter the program without restarting it. This is also true of programs which alter the interrupt RAM vectors ($200-$224) because OMNIMON! restores them to their original values.

There are a couple of small problems with using SELECT/RESET (instead of OPTION/RESET) to interrupt DOS or BASIC. OMNIMON! makes use of the SIO interrupt routines in the OS ROM by altering the interrupt vectors at $20A-$20D. This is so the printer and disk interface of OMNIMON! will work even if DOS is not in memory. Now if you return back to DOS with START/RETURN these interrupt vectors will remain in effect. But DOS hangs up occasionally unless it is using its own special SIO handlers. If you wish DOS to restore its special vectors, exit OMNIMON! with SYSTEM RESET. Another problem with SELECT/RESET is that MEMLO ($2E7) gets restored to $700 so that the FMS or any other program in low memory is unprotected. For that reason it is best to exit OMNIMON! back to BASIC with RESET. Alternatively, always use OPTION/RESET to enter OMNIMON! from BASIC.

Another way to enter OMNIMON! is particularly useful for debugging assembly language programs. This is accomplished by putting 'JSR $C001' at critical points within the program. At each of these points OMNIMON! will be entered and you will have all of its facilities available for examining the intermediate results of your program. When you are ready to continue executing your program, just exit OMNIMON! with START/RETURN. There are some restrictions on this technique however, specifically special graphics and time critical I/O.

Yet another way to enter OMNIMON! is from BASIC with a 'X=USR(49152)'. In fact, this is the recommended way to enter the monitor if you have not modified the interrupt vectors at $FFFA-$FFFD as described in the OMNIMON! INSTALLATION INSTRUCTIONS. You can exit back to BASIC in the usual manner (START/RETURN).

It should also be pointed out that OMNIMON! will be entered automatically if a 6502 BRK instruction (0) is ever executed. Thus, you can set a breakpoint anywhere in your program by storing a 0. When you pop into OMNIMON! after executing a BRK instruction, you should restore the original instruction and subtract 2 from the PC. Now you can continue

executing your code when you exit OMNIMON!.

CPU REGISTERS: C

You will notice that, upon entering OMNIMON!, the 6502's internal registers are printed out with the following heading:

PC NV-BDIZC ACCUM X-REG Y-REG STACK

The meanings of these headings are self-explanatory except for 'NV-BDIZC'. These are the individual bits of the status register spelled out. Thus, this is a snapshot of the state of the CPU just prior to entering OMNIMON!. The PC (program counter) is pointing to the next instruction to be executed. The program will continue executing at this point when you leave OMNIMON! with START/RETURN.

The CPU state can be examined at any time with the CPU REGISTERS command 'C'. In addition, the CPU state can be changed by simply positioning the cursor over the value, typing the change, and hitting RETURN. The new values for the registers will be in effect when you leave OMNIMON! to resume execution of the suspended program. The only CPU register that cannot be changed directly is the stack pointer. This can be changed only by the PUSH STACK (+) and POP STACK (-) commands.

One application for the 'C' command is to GOTO anyplace in memory. This is accomplished by altering the PC to point to the address where you wish execution to resume when you press START/RETURN. Typically this might be back to DOS, whose address can usually be found by looking in location $000A (DOSVEC).

Another area of interest is the stack. Remember, the stack pointer always points to the next FREE entry. All the values between the stack pointer and $1FF are typically return addresses of nested subroutine calls. This, in fact, is a vertical cross section of the execution history of the program. This is extremely useful for finding your way around in a program you wish to modify in some way. If you wish to to locate the part of a program which is performing a certain function, just start the program executing that function and press SELECT/RESET. Because the stack is preserved with this method of entering OMNIMON!, you can tell where the program is and where it has been by noting the PC and the return addresses on the stack. Another way of locating a certain piece of code is to search ('S') for a particular address it might reference.

PUSH STACK: + byte byte ...

The PUSH STACK command is for adding bytes to the stack and thereby increasing the stack pointer (which grows downward in the 6502). These bytes will be available to the code pointed to by the PC when OMNIMON! is exited. Notice that the first byte after '+' is the first one to be pushed onto the stack.

Please note that the stack pointer displayed with the 'C' command is not the ACTUAL stack pointer while OMNIMON! is running. OMNIMON! uses the stack for its own purposes and is actually nested somewhat deeper. It is not wise to make changes directly to the stack unless you use PUSH STACK or POP STACK. Even then you must be careful not to cause the stack to overflow or underflow.

POP STACK: -

The POP STACK command takes bytes off of the stack one at a time and decreases the stack pointer (which actually increases in value). Be careful to not cause the stack to underflow.

DISASSEMBLE MEMORY: X (start addr) (stop addr)

Just as it is possible to display memory in hex or character format, it is also possible to translate 6502 machine code to assembly language. OMNIMON! does this in a handy fashion by printing out the object code along with the instruction. Once again, it is possible to change the object code (to the left of '*') by positioning the cursor, typing the change, and hitting RETURN. Another convenience is that the value at the address specified with indirect addressing modes (without regard to the index register) is printed in parentheses.

Just like the 'R' and 'D' commands, 'X' is 'persistent'. Once you have disassembled one or more instructions, you can continue disassembling simply by holding down RETURN. This will remain in effect until 'R' or 'D' are used. The disassembler can be aborted at any time by pressing the START button.

EXECUTE MEMORY: E (option/# steps)

The EXECUTE MEMORY command is actually a single step command in disguise ('S' is used for SEARCH). This command causes the instruction pointed to by the PC to be executed. Then the registers are printed out along with the NEXT instruction to be executed. If the step count was 1 (or not specified) then execution will stop. Otherwise it will continue single stepping through the code for the specified # steps. The maximum number of steps at one time is 31 for reasons soon to become clear.

While the low order 5 bits of the optional parameter are a step count, the high order 3 bits have special meaning. The MSB means 'step forever'. Thus, 'E 80' means 'step forever and print the trace to the screen'. Notice that the trace will also be echoed to the printer if it is enabled. Stepping can be aborted by pressing START.

Bit 6 of the parameter means 'don't print the trace to the screen'. However, the trace will still be output to the printer if it is enabled. Thus, 'E C0' would step forever without printing the trace to the screen. In combination with the printer this is useful for stepping through programs which use special graphics modes.

Bit 5 of the parameter means 'sample the results of every 32 instructions'. Thus, 'E E0' would step forever without printing to the screen and the trace would be output to the printer every 32nd instruction (if it is enabled). This is kind of a weird mode, but somebody may find a use for it someday.

One other nice feature of the 'E' command is that it will treat a call to the OS as a single instruction instead of stepping through all the code in the OS. OMNIMON! does this by temporarily giving up control of the CPU but intercepting it on the return from the OS. However, you should avoid stepping through CIO calls to the screen editor (E:) unless printing to the screen is disabled with bit 6. OMNIMON! considers any address above $C000 to be OS.

We have seen that the EXECUTE MEMORY command is very powerful and flexible. One restriction, however, is that it will not step through a 'SEI' instruction. If you are stepping through a program and encounter a SEI, disassemble on past it to find the 'CLI'. Just past the CLI put a temporary BRK instruction (0). Now step through the SEI. OMNIMON! will temporarily lose control of the program but will regain it when the BRK instruction is executed. Now restore the original value to the location where the BRK was set. After subtracting 2 from the PC you are ready to continue stepping.

JSR: J addr

The JSR command is a very powerful feature for executing a subroutine and returning control back to OMNIMON!. It can be used for testing out subroutines during the development of an assembly language program. With some care it can also be used to call the OS to, say, format a disk.

When you execute the 'J' command you will notice that the registers are printed out but that the subroutine is not yet executed. In fact, the 'J' command does nothing more than change the PC to the specified address and push the address of OMNIMON! on the stack to act as the return address for the subroutine. Now you are free to set up for the subroutine call by altering the registers or memory if necessary. When you are ready to actually execute the subroutine press START/RETURN. Upon return you will notice that the PC is restored to its original value but that the other registers reflect the results of the subroutine.

As an example, put a fresh disk (or one you don't mind formatting) in drive 1. With the printer disabled, store a 1 in $301, a $21 in $302, and a $80 in $303. Now execute a 'J E453' and press START/RETURN. The disk in drive 1 will be formatted and then control will be returned to OMNIMON!.

Sometimes after interrupting a program with OMNIMON!, you will not be able to restart it without reinitializing it. The start and initialization addresses for a program are typically at $000A and $000C respectively. Since a proper initialization routine is always a subroutine, you can use 'J (init addr)' to initialize the program. When control returns to OMNIMON!, you need only change the PC to the start address and exit OMNIMON! with START/RETURN to restart the program.

QUESTIONS?

I welcome comments, suggestions and dealer inquiries:

DAVID YOUNG
CDY CONSULTING
421 HANBEE
RICHARDSON, TX 75080
(214)235-2146

LIMITED WARRANTY

For a period of one year following the date of purchase CDY CONSULTING will repair or replace any OMNIMON! unit proven to be defective. Please return the defective unit to the place of purchase.

Binary Load Files

This is one area that most people are a little fuzzy on. It's not surprising really, since there does not appear to be any definitive documentation on it anywhere! An exhaustive presentation will be made here even though it will be quite short.

Definition: **load vector** - from 4 to 6 bytes consisting of 2 optional bytes of FF FF, a 2 byte start address, and a 2 byte end address (in that order).

Example: FF FF 00 06 02 06 - The first 2 bytes are optional and ignored during the load process. The second 2 bytes are a start address of $600 and the last 2 bytes are an end address of $602.

The only time that the first 2 bytes of FF FF are required is at the beginning of a binary load file. If those bytes are not there, DOS will refuse to perform the binary load. (The 'G' command of OMNIMONL will, however, load it gladly.) The rest of the time the first 2 bytes of FF FF, if they exist, are ignored. The only time that these 2 optional bytes should occur anywhere else but at the beginning of a file is if 2 binary load files were appended together.

What does a load vector do? It tells DOS (or the 'G' command of OMNIMON) where in memory to put the 1 or more bytes which follow in the file. How many bytes is determined by subtracting the start address from the end address and adding 1. In the example above, 3 bytes would be read from the file and put in locations $600 to $602.

What happens when enough bytes have been read in to satisfy a load vector? These things will happen in this order:

1) Locations $2E2 and $2E3 will be examined. If they are both zero, goto step 2. If they are nonzero, a JSR will be made to the address contained in these locations. Upon return, zero $2E2 and $2E3 and fall into step 2.
2) If the end of file is not reached (i.e., there are more bytes in the file), another load vector is assumed to immediately follow and will be processed as previously described.
3) If the end of file (EOF) is reached, examine locations $2E0 and $2E1. If they are zero, terminate the binary load. If they are nonzero, do a JSR to the address in these locations. Upon return, terminate the load.

Still confused? Let me try to simplify. If you see a load vector like 'E2 02 E3 02', you know that the subroutine at the address specified in the following 2 bytes will get executed immediately, prior to continuing the load process. If you see a load vector like 'E0 02 E1 02', you know that the subroutine at the address in the following to 2 bytes will be executed after the end of file is reached.

Now that you understand everything there is to know about binary load files, let's take a typical example: converting the BASIC cartridge to binary load file. I use this example because it is instructive and, because of the lack of copyright notice, appears to be legal. Using this technique on other cartridges could be illegal and may not work anyway due to the booby traps designed to prevent them from running out of RAM.

Whenever attempting something new, it is advisable to try it out manually from OMNIMON first whenever possible. Let's see what it takes to get the BASIC program to run out of RAM:

- -Turn on the computer with BASIC installed and pop into OMNIMON.
- -Insert a formatted scratch disk into the drive and execute the following command: 'W1 A000 40 (RETURN)'.
- -Remove the BASIC cartridge, boot up DOS and pop into OMNIMON.
- -Because OMNIMON restores MEMLO to $700, we should reinitialize DOS by doing a JSR to the address at DOSINI ($C,D). For 2.0S that would be 'J1540 (RETURN) (START/RETURN)'.
- -Move the screen down by storing a $90 in location $6A and doing a JSR $F3F6: 'A 6A 90 (RETURN)','J F3F6 (RETURN) (START/RETURN)'
- -Read BASIC back into memory with 'R1 A000 40 (RETURN)'.
- -Find the initialization address by looking at location $BFFE. Since that is $BFF9, execute 'J BFF9 (RETURN) (START/RETURN)'.
- -Find the start address by looking at location $BFFA. Since that is $A000, execute 'J A000 (RETURN) (START/RETURN)'.

BASIC will now come up running. Now we want to create a binary load file to simulate the last 4 steps:

- -Type 'DOS' to get to the DOS menu.
- -Use the 'K' command to save BASIC as a binary load file giving it the start address of A000 and the end address of BFFF.
- -Pop into OMNIMON and put it in linked mode. Read the first sector of the new file into $6000 (see top of page 7 if you don't know how to find the first sector of a file).
- -Hold down RETURN until 'EOF' is printed out. You now have the last sector of the file in memory.
- -Determine the last byte of that sector in use by looking at the byte count ($607F). Start adding the following bytes at location $6000 + byte count (we are appending to the file):6A 00 6A 00 90 E2 02 E3 02 F6 F3 00 98 08 98 20 06 98 6C FA BF 6C FE BF E0 02 E1 02 00 98.
- -Increase the byte count ($607F) by $1E and write that sector back out to the disk with 'W (RETURN)'.

Here is an explanation of these load vectors: The first 11 bytes move the screen down and the rest load a little routine into $9800 to initialize and start the cartridge. Disassemble them to see how. Except for the booby traps, you should be able to easily extend this technique to 16K cartridges. However, some cartridges assume that memory is clear, so you can append the following load vectors: 09 98 23 98 A0 00 A9 07 85 D5 A9 00 85 D4 A9 00 91 D4 E6 D4 D0 F8 E6 D5 A9 30 C5 D5 D0 F0 E2 02 E3 02 09 98 60. This will clear out DOS before starting the cartridge.

OMNIMON! INSTALLATION INSTRUCTIONS

NOTE: If you purchased OMNIMON! installed in the OS piggy-back board, ignore this sheet. Instead, refer to the installation instructions provided with the board.

The OMNIMON! program comes in a 2732 compatible ROM. This can be installed only in special hardware such as RAMROD MMOS by NEWELL INDUSTRIES which accepts a 2732 in the $C000 block of memory. Follow the instructions provided with this hardware for installation of the chip.

To make sure the installation is correct up to this point, power up the computer. If everything appears to be normal, try entering OMNIMON! from BASIC with a 'X=USR(49152)'. If it fails this test then double check the installation of the chip. If this works properly then proceed with the next paragraph.

For proper operation of OMNIMON! the following locations should be changed in the OS (assuming your OS is in EPROMS):

$FFFA-$FFFD - $CE,$CF,$F5,$CF

This allows the SELECT and OPTION keys to work in conjunction with SYSTEM RESET for entering OMNIMON!. If If the OS chips were provided with the RAMROD then these changes have already been made. If you do not make this mod then it is possible to enter OMNIMON! from BASIC with 'X=USR(49152)'. However, this is less convenient than the SELECT/RESET or OPTION/RESET technique, which interrupts any program which may be running.

If the system fails to power up after you have made these mods then the mods were not done correctly. When done correctly the system should power up normally and you should be able to enter OMNIMON! with SELECT/RESET or OPTION/RESET. Installation of OMNIMON! is now complete.

OMNIMONA!
For Advanced Users

By deleting the HELP command and shortening the user messages, room was freed up for a few more desirable commands and useful features. This document describes these additional features. Please refer to the OMNIMON! USER'S MANUAL for a description of the standard version of the monitor. A copy of OMNIMONA! can be acquired by sending a 350nS 2732 EPROM + $2.00 postage or $10.00 to CDY Consulting, 421 Hanbee, Richardson TX 75080. OMNIMONA! can be freely copied for dissemination among the community of OMNIMON! users.

1) Hex converter: H #

   # can be either in hex or decimal. Decimal #'s are terminated with a non-hex character. For example, 'H100T' yields '$64 = 100'.

2) Verify memory: V addr#1 addr#2 #bytes

   Compare the specified # bytes starting at the two blocks of memory specified by addr#1 and addr#2. Print the differences.

3) Happy upload and download:

   If the disk is read or written to with a sector # of $800 or greater, the Happy drive treats the sector # as an internal buffer address ($800-$13FF). On multiple reads or writes the sector # is incremented by $80 each sector. This is not only useful for programming the Happy drive but is also handy for storing stuff on a temporary basis.

4) RESET mod:

   There is support for the hardware mod which hooks the SYSTEM RESET switch into the RESET line of the CPU. This is accomplished by inserting a 47 ohm resistor into the pads provided on the motherboard near CR103 and R183. A momentary switch to connect the resistor is also needed unless you wish to do a coldstart everytime SYSTEM RESET is pressed. With this mod and OMNIMONA! it is possible to recover from a system lockup. This is accomplished by pressing the momentary switch and pressing SELECT/RESET. This will pop into the monitor with the PC reflecting the instruction the CPU was trying to execute when it locked up (probably an illegal one). At that point RESET can be pushed by itself to do a normal warmstart.

5) P command enhancement:

   The trace turned on by the P command can now be redirected to any output device. If it is desired to output to something other than the printer, store the device specification someplace in memory and point $105 to that location. The P command will then open up an I/O channel to that device and the next P command will close it. For example, if you were to store 'D:TEMP ' (notice the blank used as a terminator) at $600 and store 00 06 at $105, the P command would open up a disk file. This feature is especially useful in conjunction with the disassembler or the single-step.

OMNIMON! UPDATE: ZAXXON PATCH

The growing number of enthusiastic OMNIMON'ers I hear from everyday give credence to my belief that OMNIMON! is fast becoming the de facto monitor for the ATARI 400/800. It joins a growing number of products which result in an enhanced operating system (e.g., FASTCHIP, RAMROD, MPP-1100, etc).

Now, what if I were to tell you that there is a well known, respectable software publisher that is deliberately trying to be incompatible with all users except those with standard ATARI OS hardware? Who (except maybe ATARI) would be so silly as to pull a stunt like that? Yes folks, it seems that Datasoft Inc. has decided to publish the new disk version of ZAXXON with a protection mechanism so clever that it runs only with standard ATARI, plain vanilla OSA or OSB with the sloth-like ATARI floating point chip. Evidently Datasoft also thought it would be cute to keep the unsuspecting buyer unaware of his predicament until he got home and discovered that his proud new purchase would not run on his machine until he ripped out all of his expensive non-ATARI OS hardware, to which he has become very attached. Well, perhaps the person in charge of coming up with a protection scheme (that must be quite fun really!) decided that it would heighten the player's anticipation if it were a real challenge to get it to run on the system in the first place. Now that would be a novel idea! Promote a new game with a media blitz for months prior to release and then incorporate a protection scheme so clever, so delightfully devious that it will not run on ANYONE's system, even if all the equipment (down to the 825 printer and 830 modem) were ATARI. The initial challenge of this ingenious game would be to break it so that it would run at all!

But let's get to the matter at hand: how to get ZAXXON to run on our enhanced OS's. It turns out that there is a very simple 3 byte patch to the ZAXXON disk which does the trick:

```
Sector $17D - bytes $26-$28
          From: 23 C0 1A
          To  : 43 EB 3A
```

In OMNIMONese this translates to:

```
R 17D 6000
A 6026 43 C0 3A
W 17D 6000
```

So this should keep everyone satisfied unless Datasoft decides to keep us on our toes by shuffling their protection around every so often. And what about other programs which might come out with similar, if just as inane, protection schemes? If someone will make me aware of it, I will enjoy coming up with a patch to make it run.

David Young
CDY Consulting
(214)235-2146